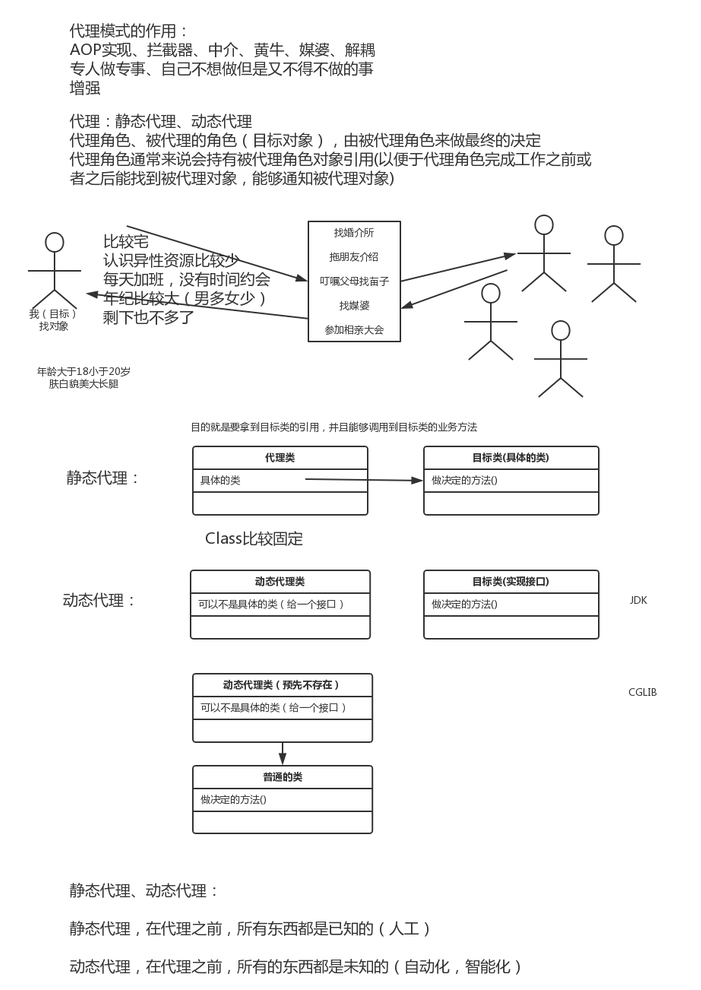
# 设计模式

## 原型模式

## 代理模式



### 静态代理

直接引入对象，若目标对象方法增加，需要调整代理对象，关联性太强

### 动态代理

#### JDKProxy

实现原理：

1. 获取目标对象的引用， 通过反射获取目标对象所有方法
2. JDK Proxy类新创建一个类(类似$Proxy0)，同时该类需要实现目标对象的接口所有方法
3. 动态生成Java代码，通过InvokeHandler调用目标对象方法
4. 编译代理类
5. 加载代理类

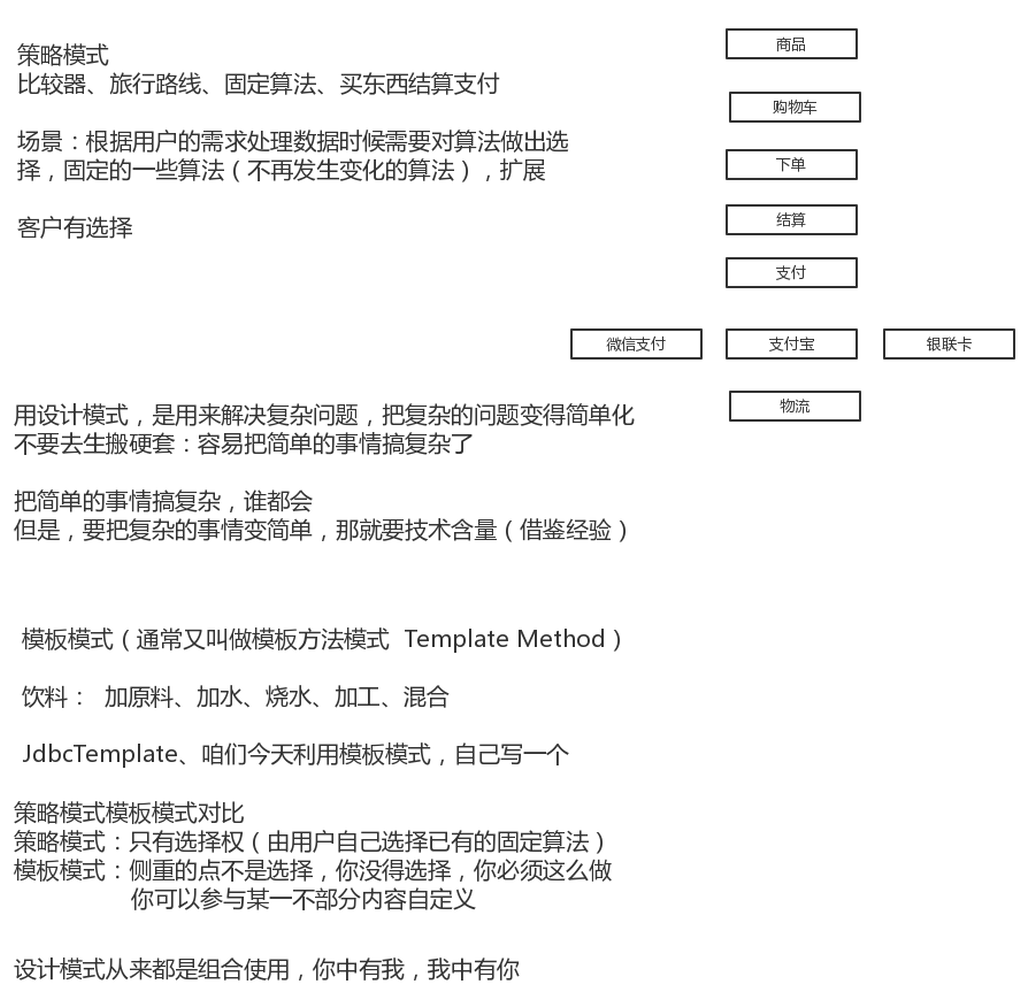
#### **CGLibProxy**

## 策略模式

场景： 根据用户的需求处理数据时，选择相应固定的算法（不在发生变化的算法）

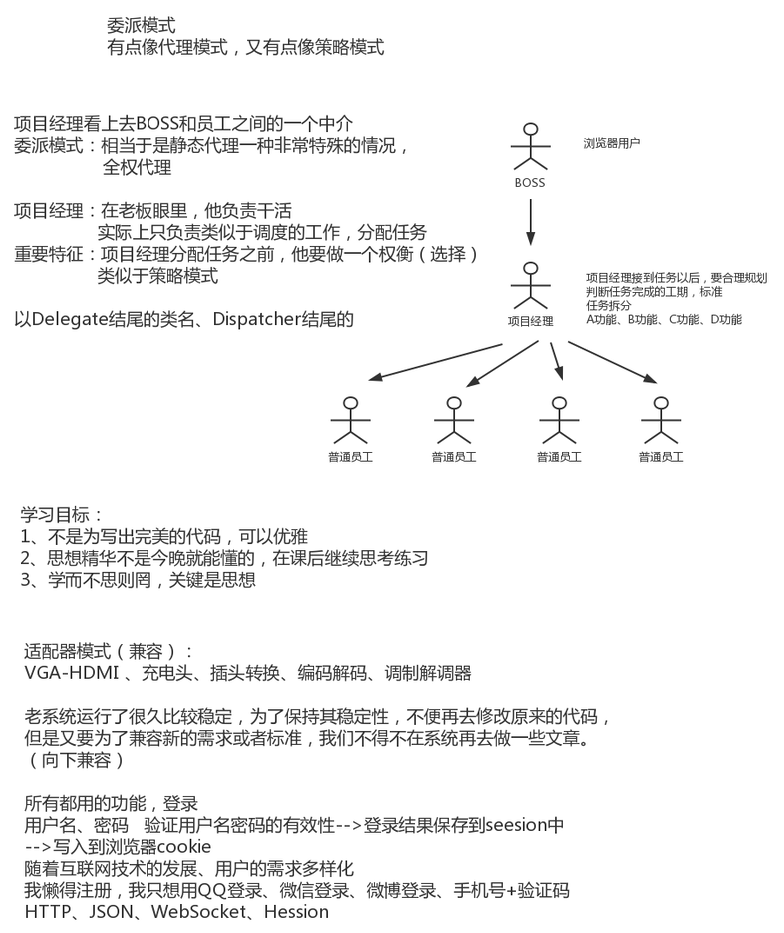
## 模板模式

通过抽象类提供抽象方法，即提供模板方法



## 委派模式

此模式不属于23种模式之中，在Spring中常以Delegate或者Dispatcher等结尾的，都是委派类。



## 适配器模式

## 装饰者模式

在原类基础上增强，本身仍属于原类型，是一种特殊的适配器模式

## 观察者模式

## 总结

### 关联

1. AOP 面向切面编程，利用动态代理的技术实现面向切面编程的思想，模块与模块之间需要遵循一定的规则才能组合，即面向规则编程。常用用于事务管理、日志管理、权限认证、上下文处理以及异常捕获
2. OOP 面向对象编程
3. BOP 面向Bean编程
4. IOC 控制反转，创建对象的控制权反转，所有的对象的创建，都由Spring创建，相对非Spring，谁使用谁创建的发生了变化。
5. DI（依赖注入）技术，解决对象属性赋值，动态调用getter、setter（基于反射）

# Spring源码分析

Sping是一种轻量级、面向Bean、松耦合、万能胶（与其他框架无缝集成）、设计模式运用灵活的业务框架。

## 2.1 Spring 架构设计

### 2.1.1 IOC体系结构

1. Bean初始化过程：定位→加载→注册→依赖注入